

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

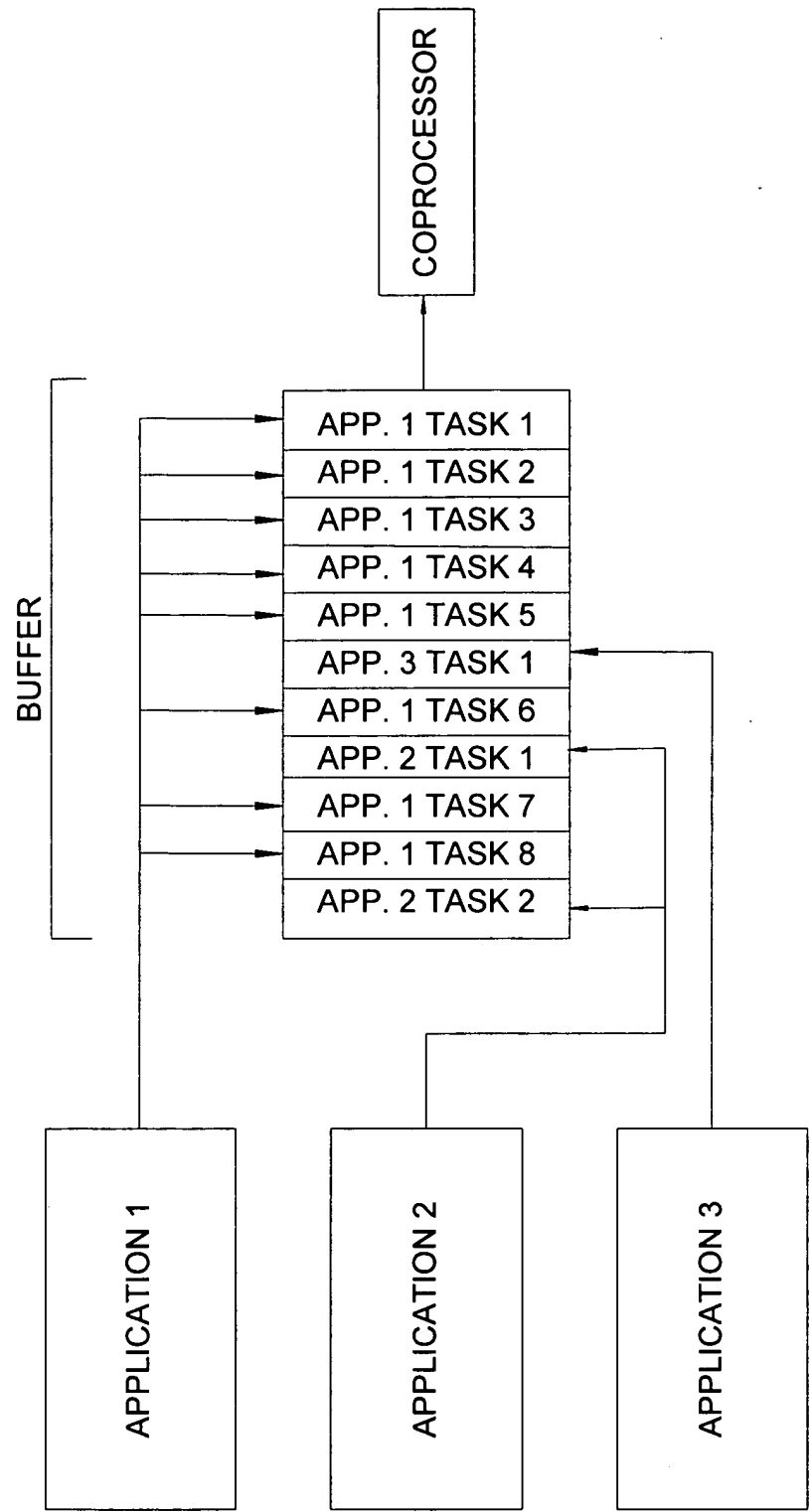


FIGURE 1
(prior art)

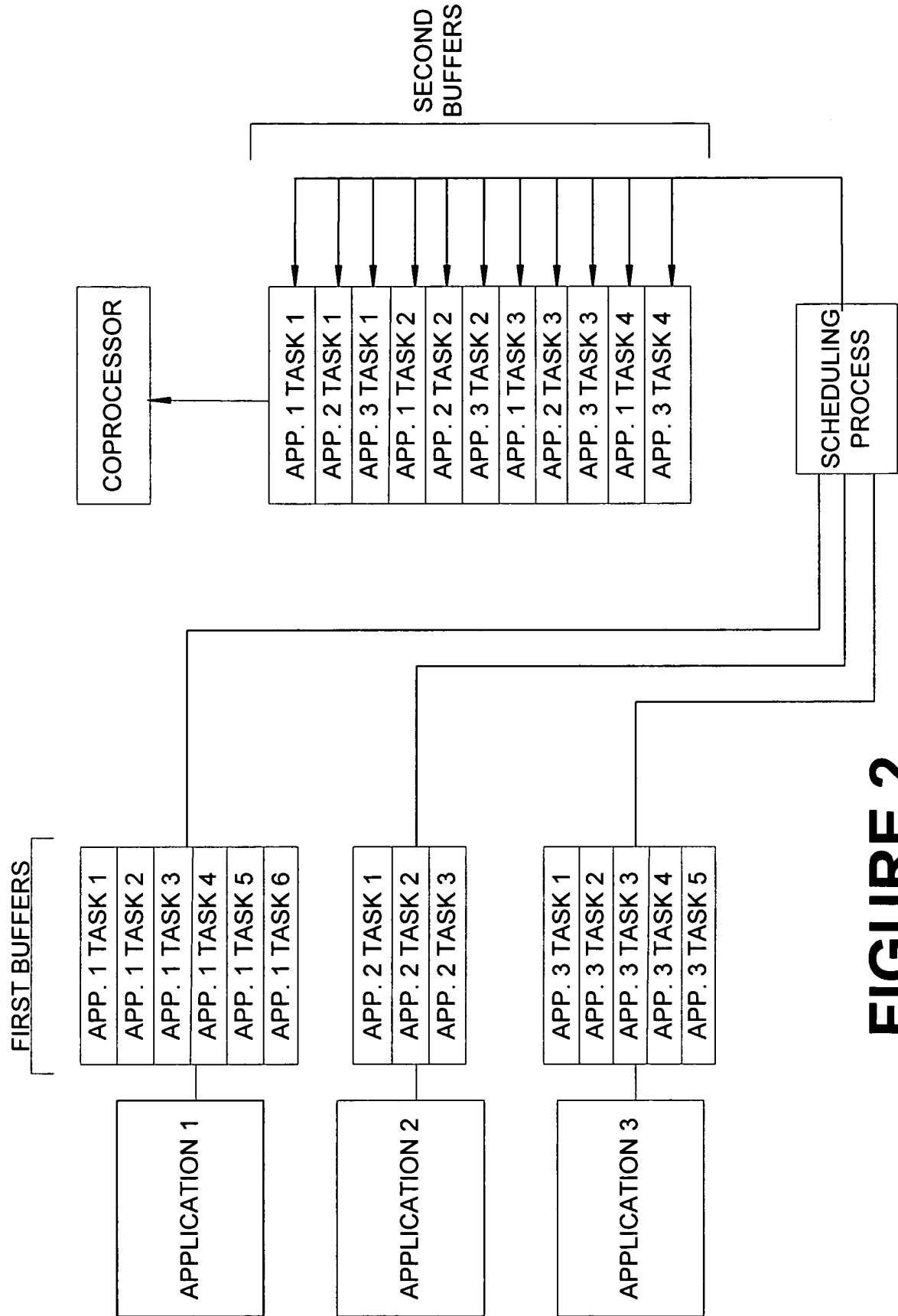


FIGURE 2

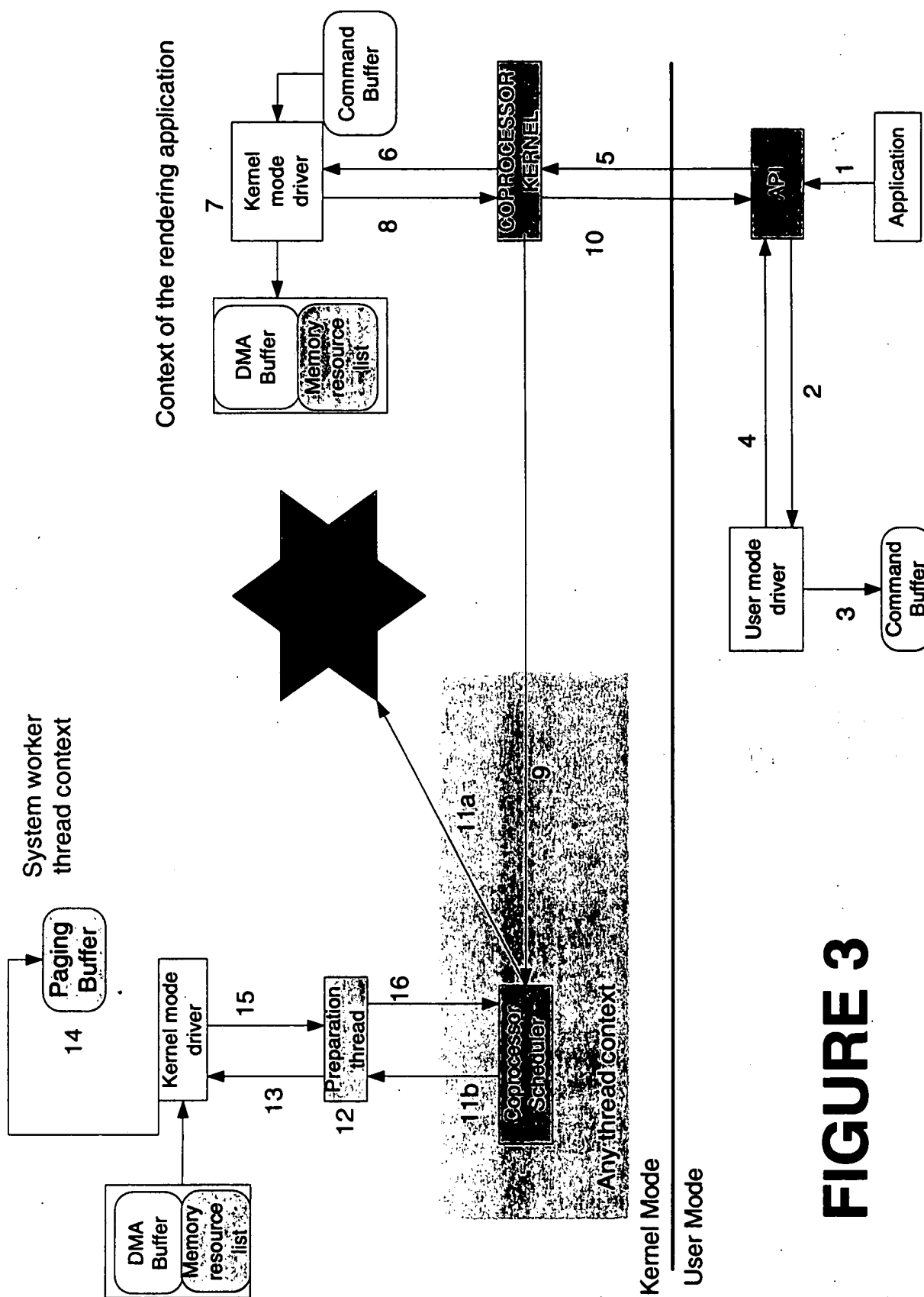
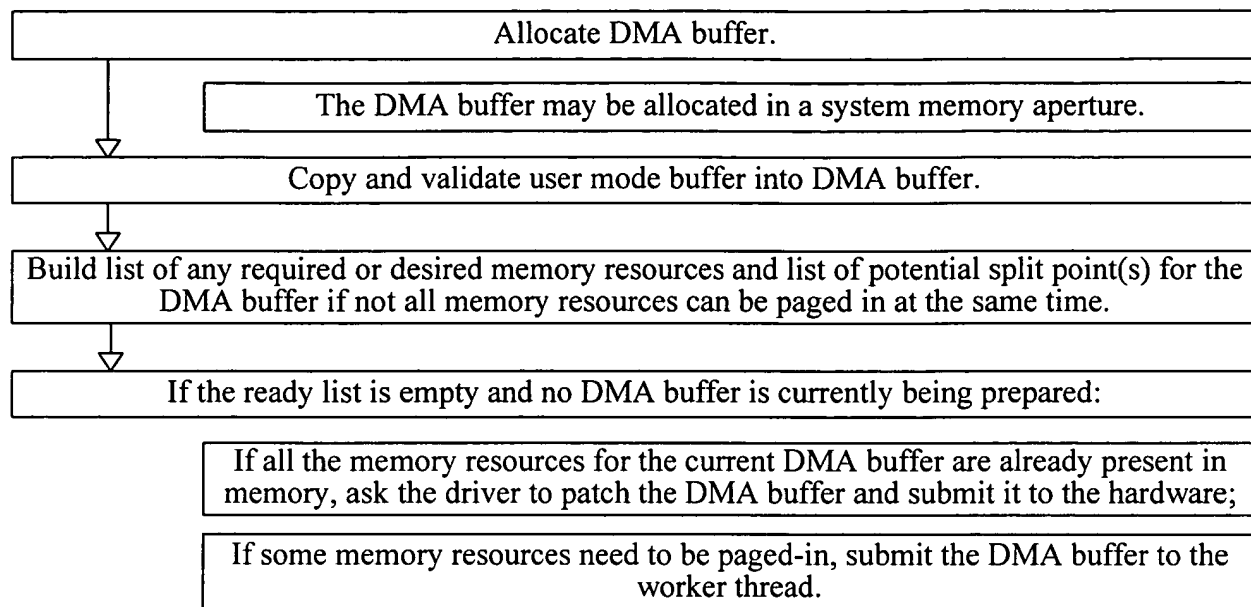


FIGURE 3

Exemplary algorithm

PROCESS A: Submit (irq passive, rendering thread context)



PROCESS B: Quantum expires (irq device, any thread context)

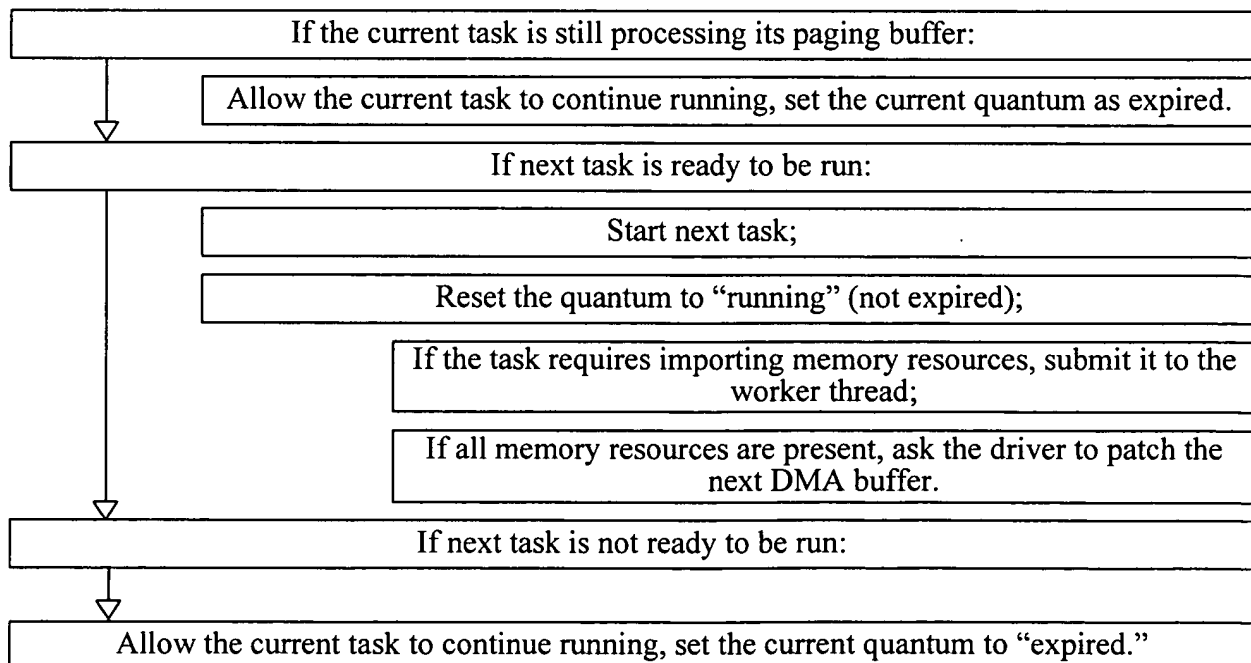


FIGURE 4(A)

Exemplary algorithm

PROCESS C: Task finishes (irq device, any thread context)

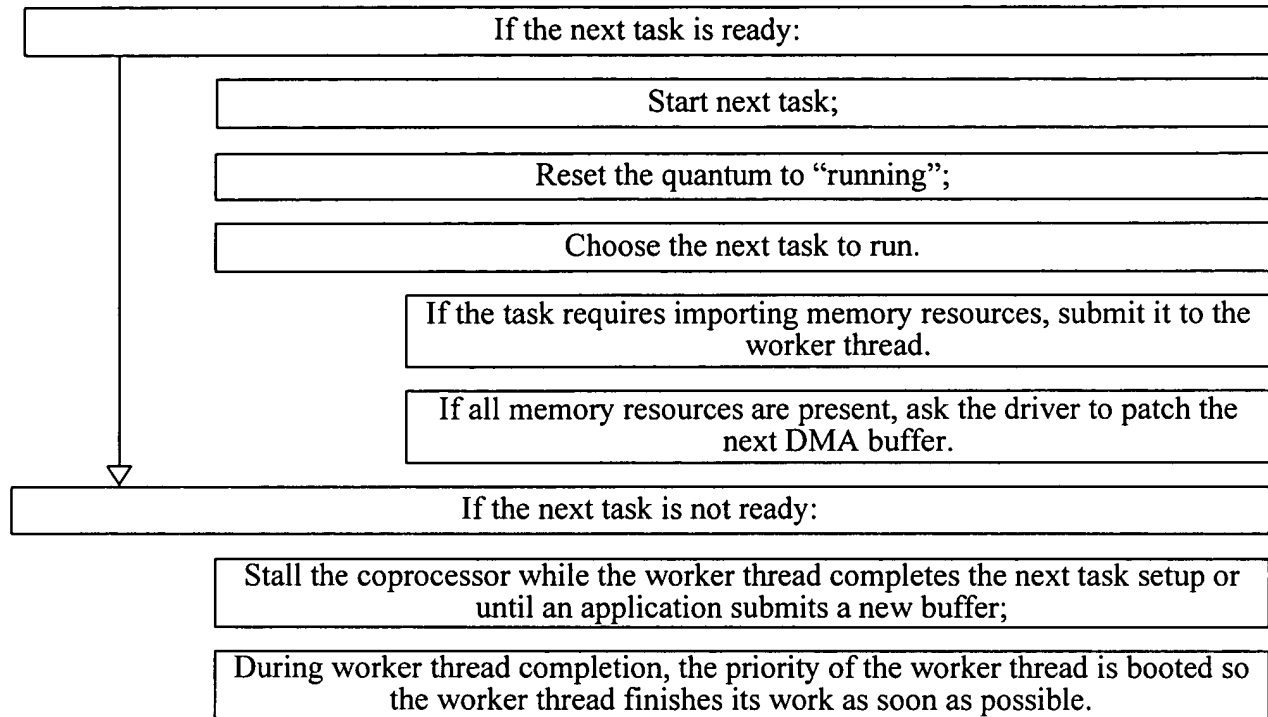
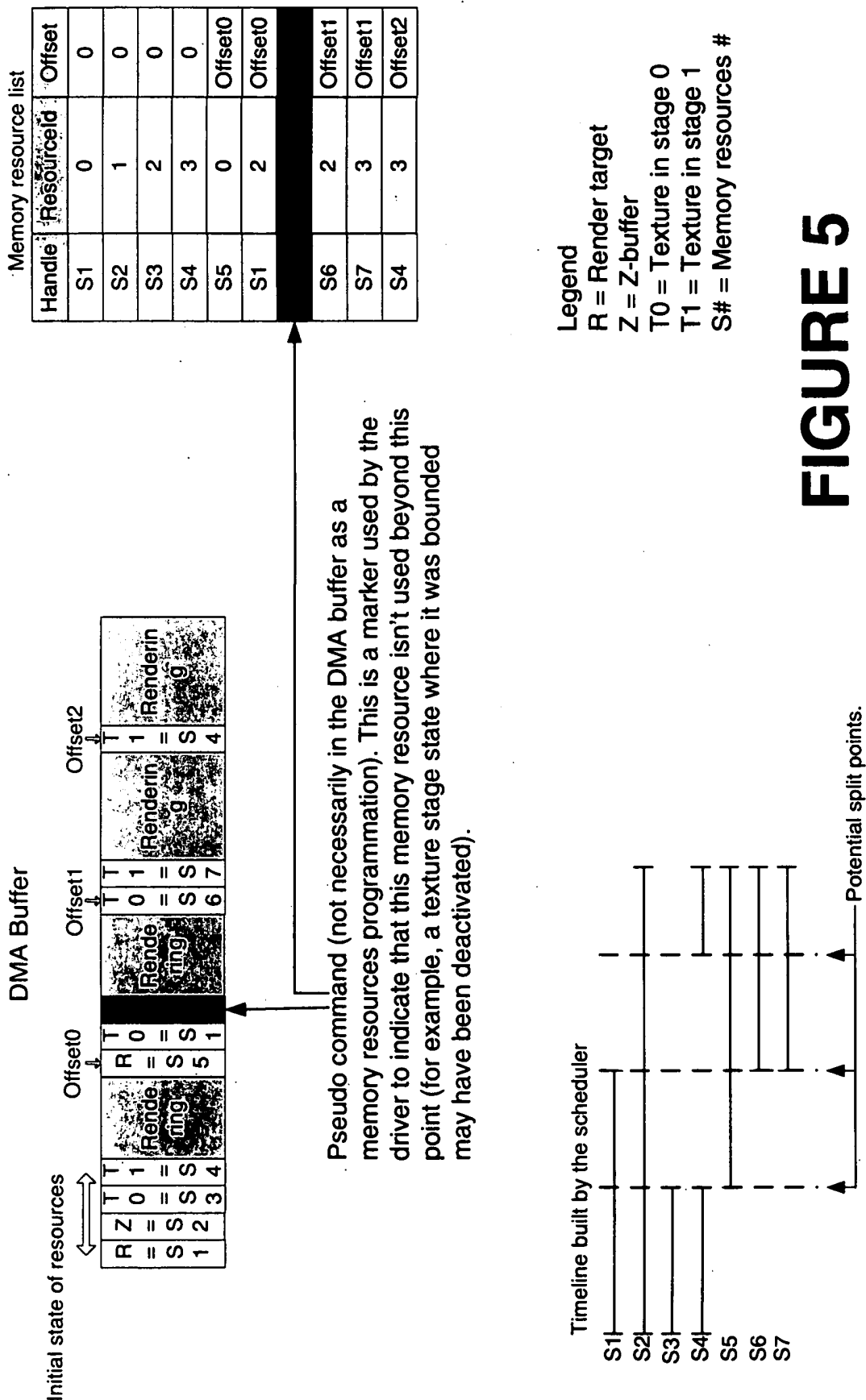


FIGURE 4(B)



Exemplary algorithm

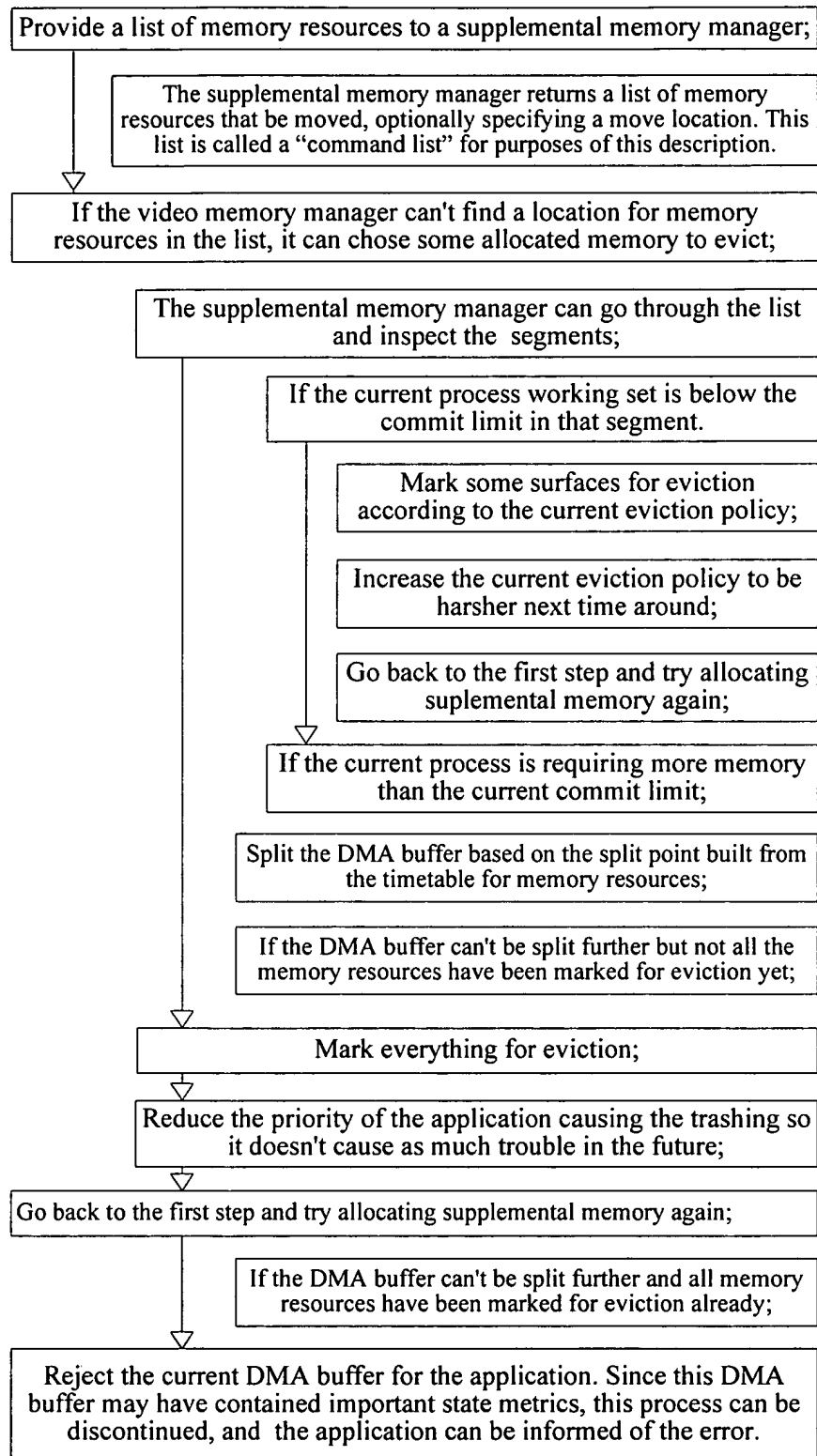


FIGURE 6

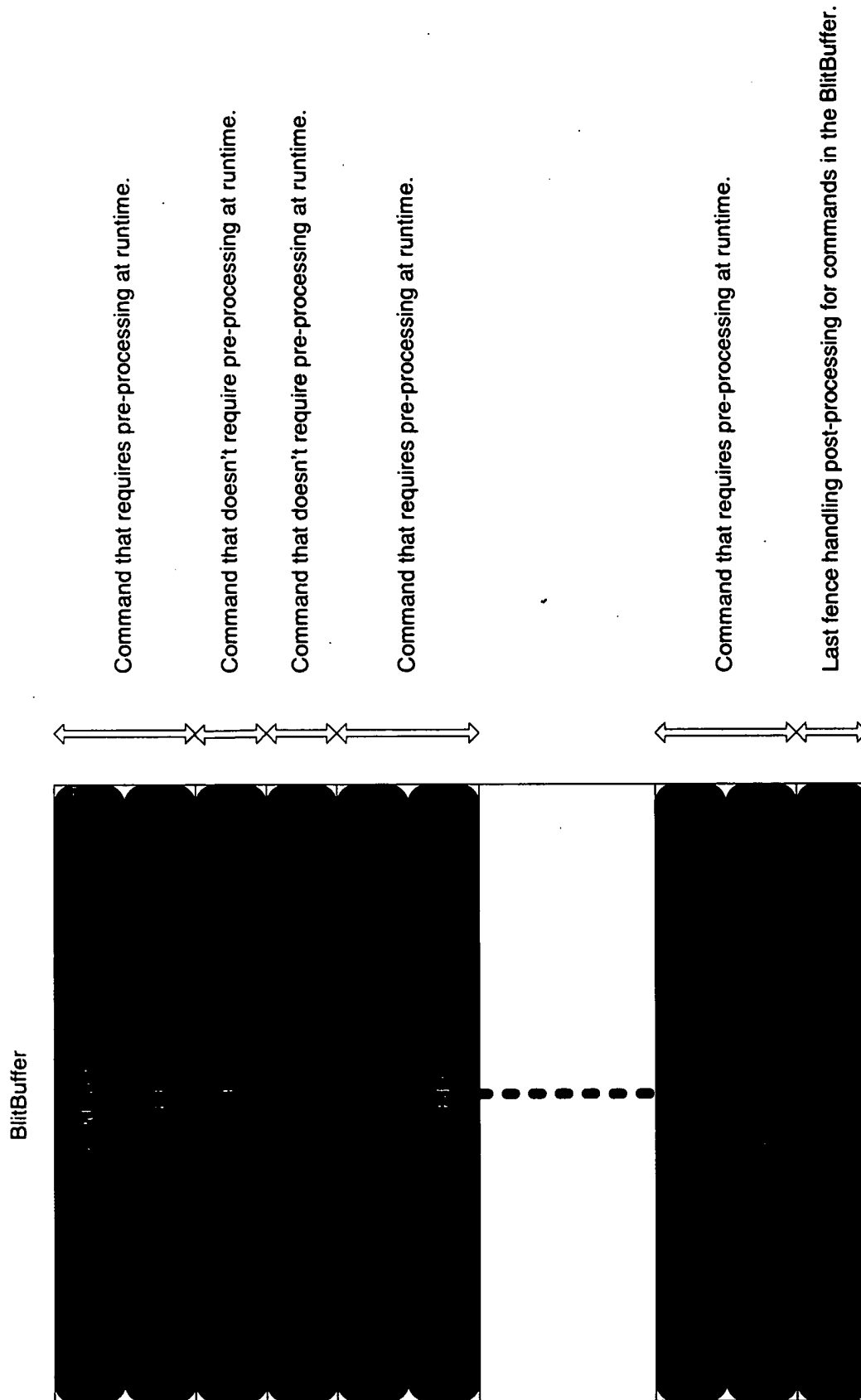


FIGURE 7

Exemplary algorithm

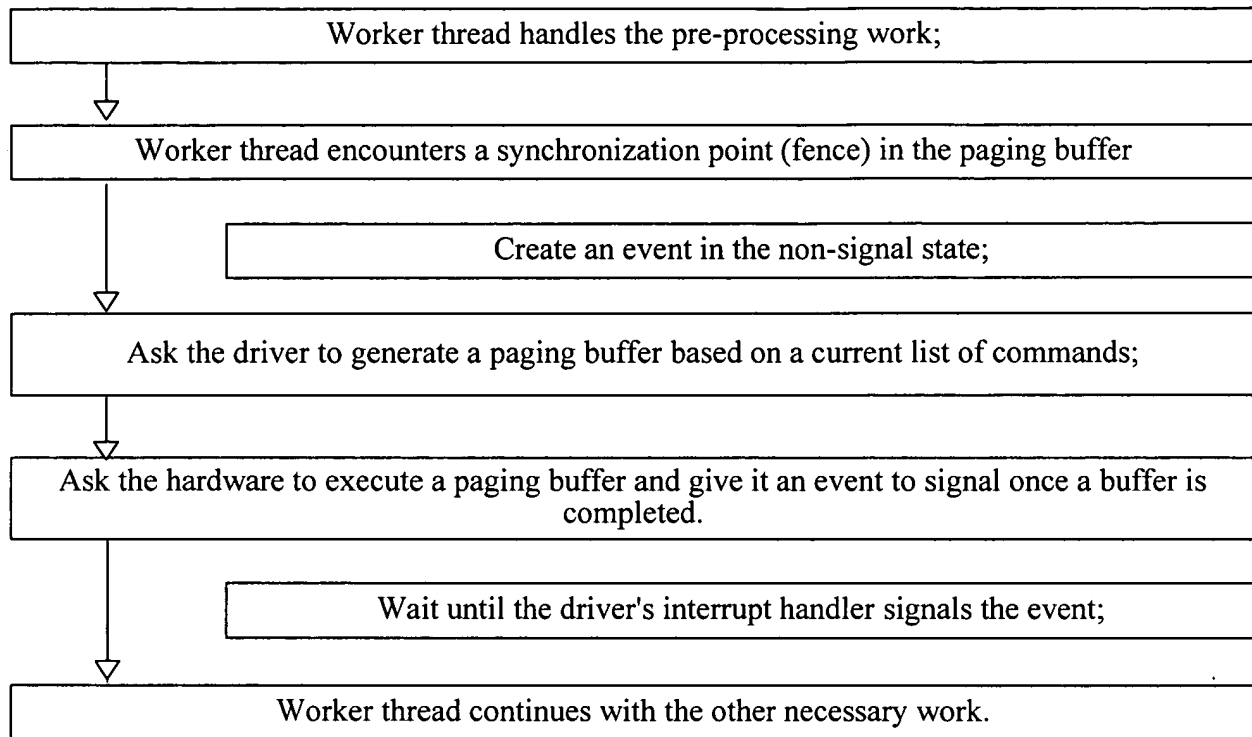


FIGURE 8

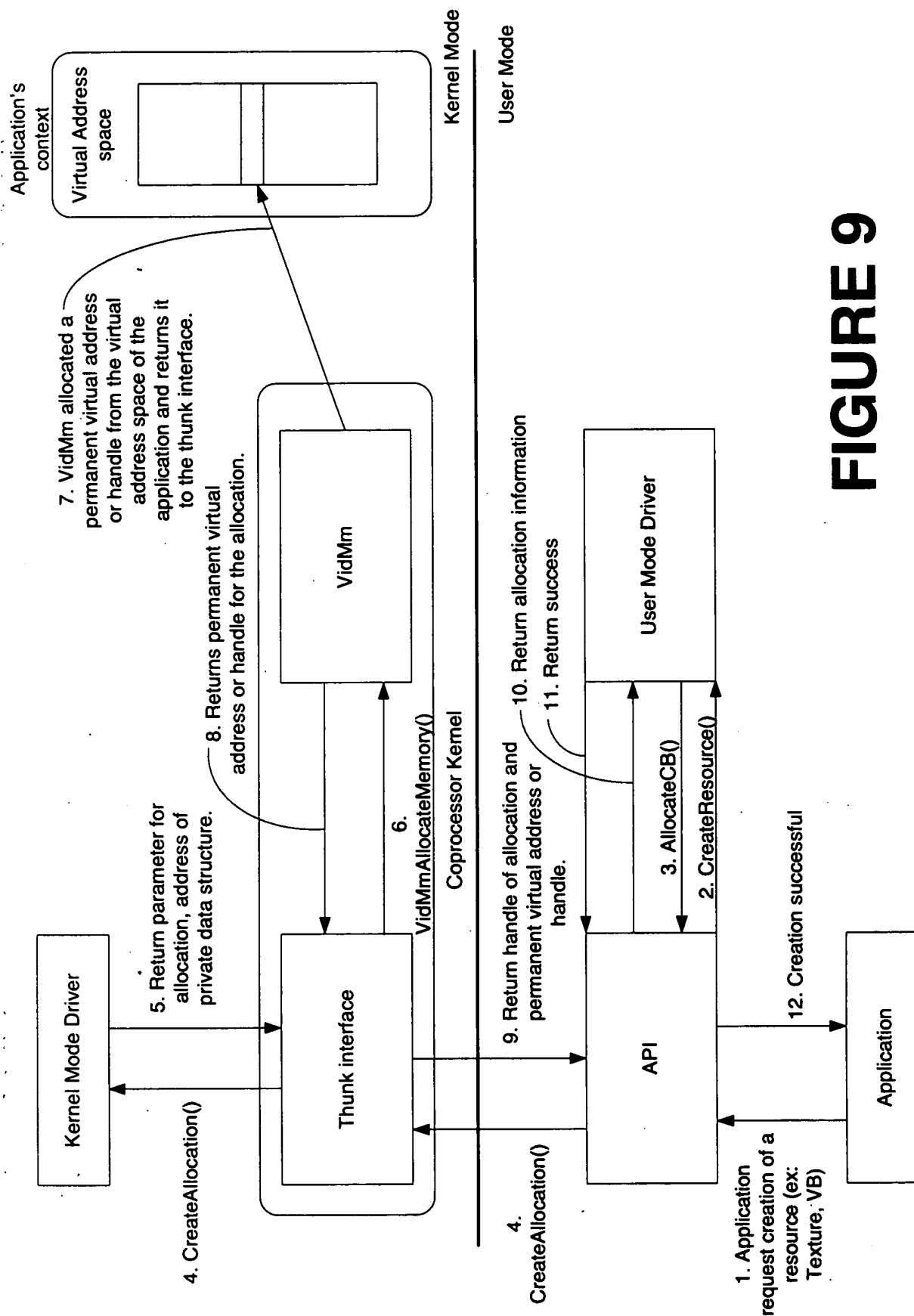


FIGURE 9

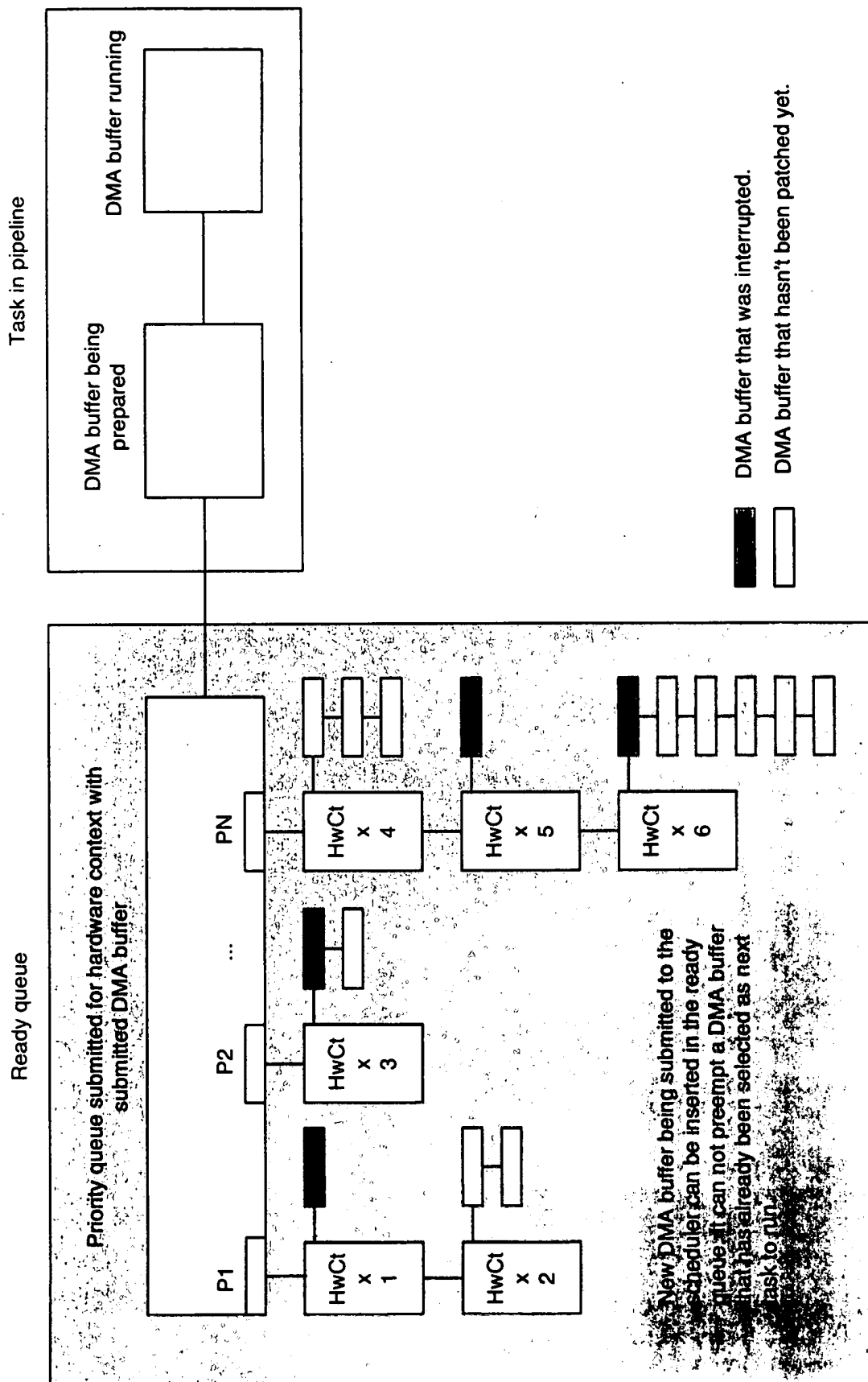


FIGURE 10

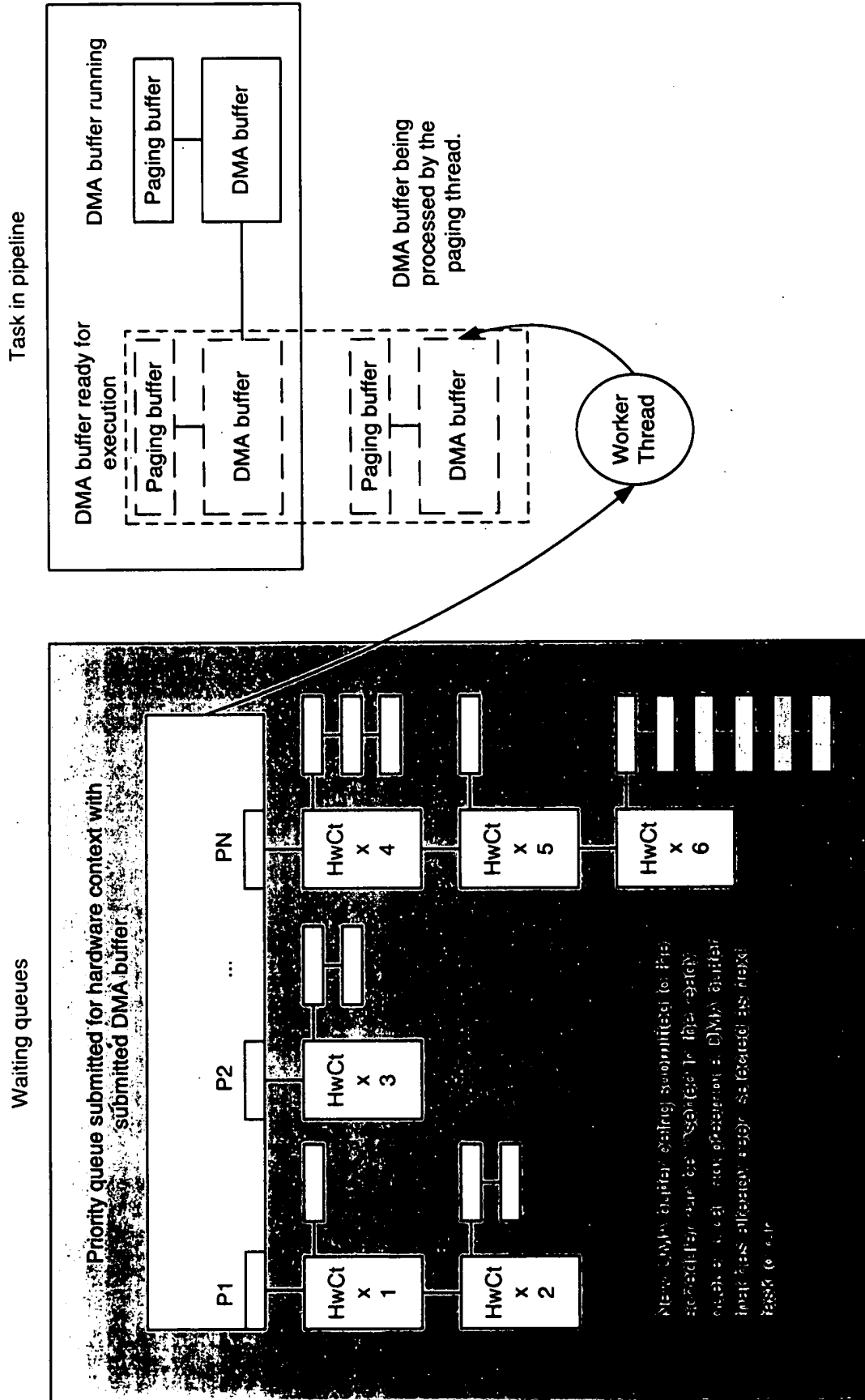


FIGURE 11

PROCESS A: Submit (IRQL passive, rendering thread context)

If no DMA buffer is being prepared or is ready for execution.

If all the memory resources for the current DMA buffer are already present in memory,

If the coprocessor is idle, give the DMA buffer to the coprocessor.

Else insert the DMA buffer in the ready-to-execute slot.

If some memory resources need to be paged in, submit the DMA buffer to the paging thread.

Else, insert the DMA buffer at the end of the list for the current context.

PROCESS B: Quantum expires (IRQL device, any thread context)

If the current task is still processing its paging buffer,

Allow the current task to continue running.

Set the current quantum as expired.

Else, if next DMA buffer is ready to be run,

Reset the current priority of the current context to its base priority.

Move the current context to the end of the queue for its priority.

Submit next DMA buffer to the coprocessor.

Reset the quantum as being running (not expired).

Choose the next DMA buffer to execute.

If the DMA buffer requires paging, submit it to the paging thread.

Else, all memory resources are already present; just insert the DMA buffer in the ready slot.

Else, the next task isn't ready to be run;

Allow the current task to continue running.

Set the current quantum as expired.

PROCESS C: Task finishes (IRQL device, any thread context)

If next DMA buffer is ready to be run,

Submit next DMA buffer to the coprocessor.

Reset the quantum as being running (not expired).

Choose the next DMA buffer to execute.

If the DMA buffer requires paging, submit it to the paging thread.

Else, all memory resources are already present; just insert the DMA buffer in the ready slot.

Else, the next task isn't ready;

If the paging thread is currently working on the next DMA buffer, boost the priority of the worker thread temporarily so it finishes its work as soon as possible.

FIGURE 12(A)

PROCESS D: Paging thread (IRQ_L passive, system thread)

Set current eviction policy to first policy.
Ask the memory manager to page in the resource list.
If all the resource were paged in successfully,
Move the paging buffer and DMA buffer to the ready-to-execute slot.
If the quantum of the current DMA buffer is expired
Submit next DMA buffer to the coprocessor.
Reset the quantum as being running (not expired).
Choose the next DMA buffer to execute.
If the DMA buffer requires paging, submit it to the paging thread.
Else, all memory resources are already present, just insert the DMA buffer in the ready slot;
Else if the memory manager failed because the paging buffer is full
Wait until the current DMA buffer's quantum end or finishes.
Submit the paging buffer to the coprocessor.
Wait until the paging buffer is done.
Go back asking the memory manager to paged-in the remaining of the resource list.
Else if the memory manager failed because there isn't enough available resource
If we've passed the last eviction policy
Undo the resource move, or run the paging buffer.
Reject the DMA buffer.
We're done.
Else if the current eviction policy is above application interference.
If the DMA buffer hasn't been split yet.
Split the DMA buffer at the closest point to the current paged-in resources.
If no more resources are needed
Move the paging buffer and split DMA buffer to the ready-to-execute slot.
Move the remaining DMA buffer back to the head of the ready queue for the context.
If the quantum of the current DMA buffer is expired,
Submit next DMA buffer to the coprocessor.
Reset the quantum as being running (not expired).
Choose the next DMA buffer to execute.
If the DMA buffer requires paging, submit it to the paging thread.
Else, all memory resources are already present; just insert the DMA buffer in the ready slot;
Ask VidMm to mark candidate for eviction using the current policy.
If VidMm returns an error saying no memory could be marked with the current policy,
Increase the eviction policy.
Go back to the start of the eviction policy check.
Else, some memory was marked.
Go back to trying to page in the resources.

FIGURE 12(B)

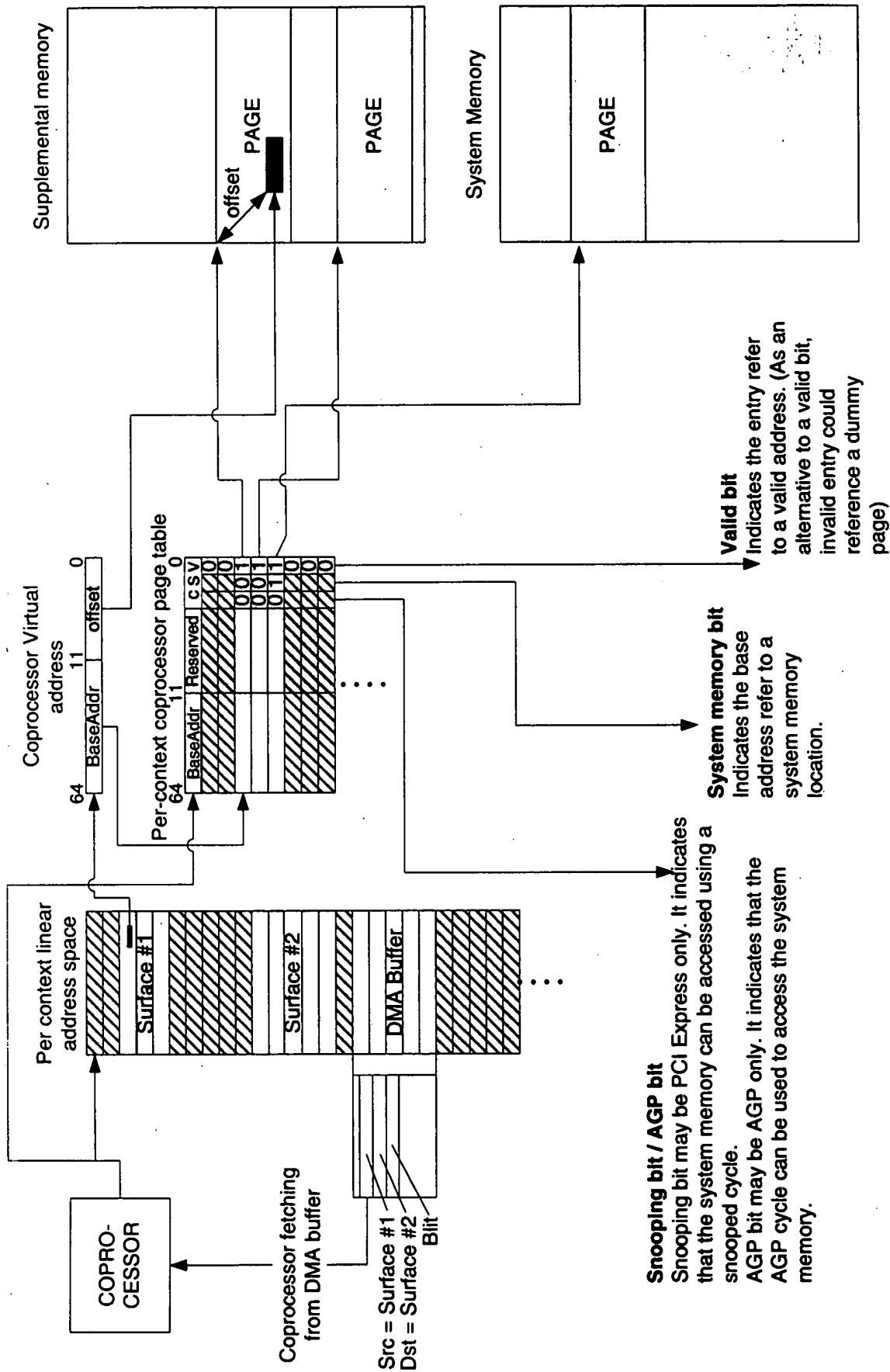


FIGURE 13

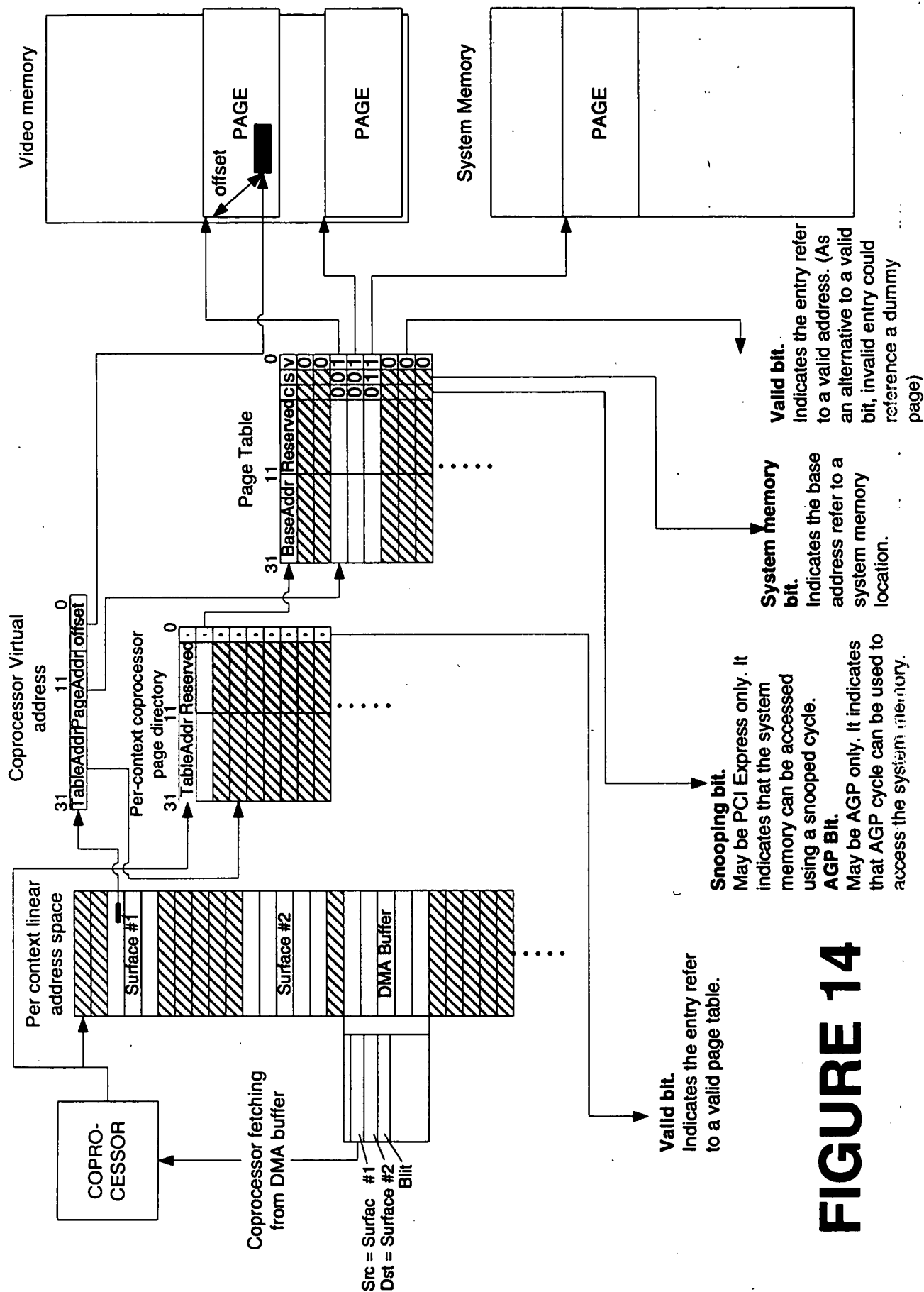


FIGURE 14

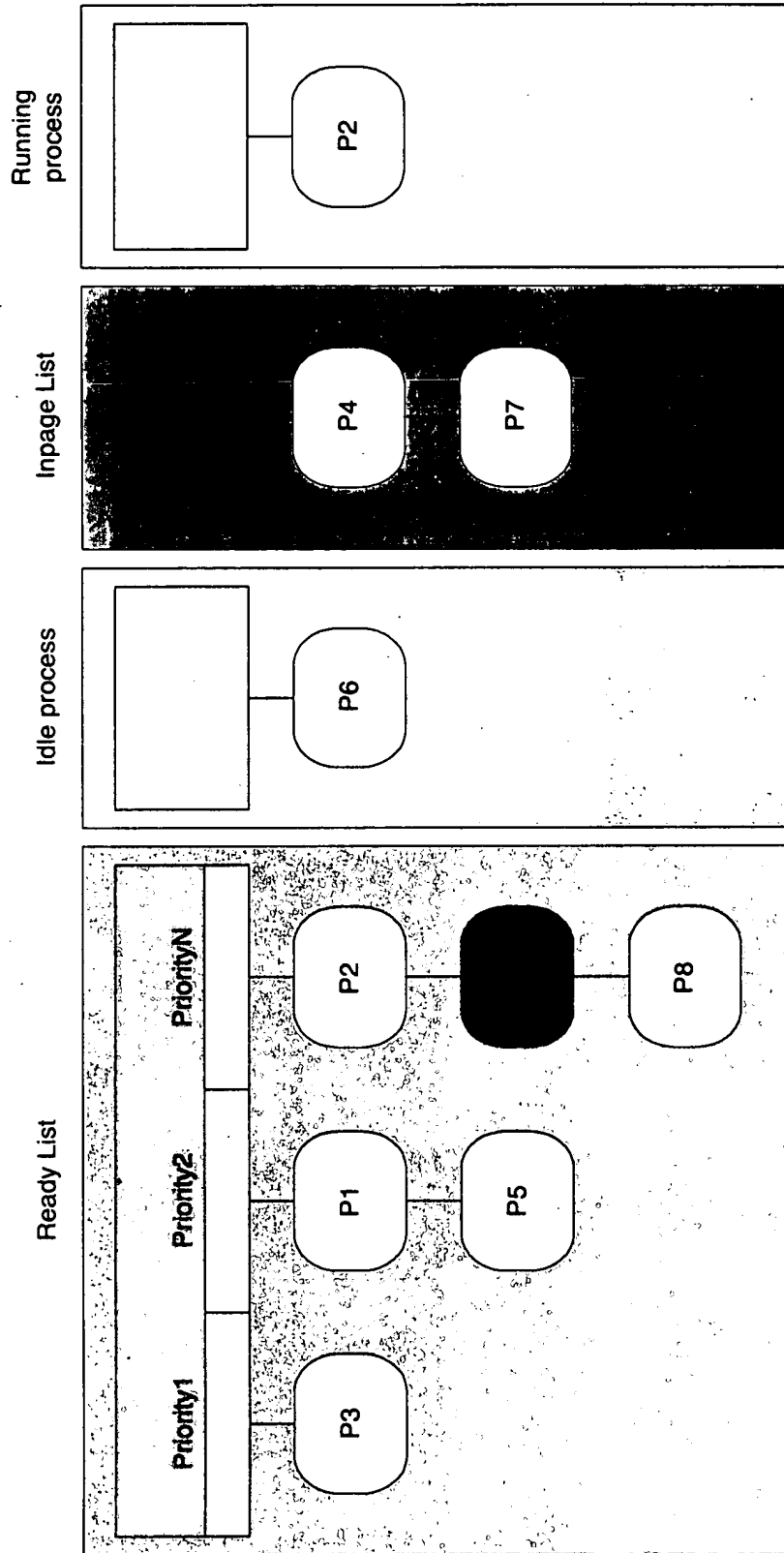
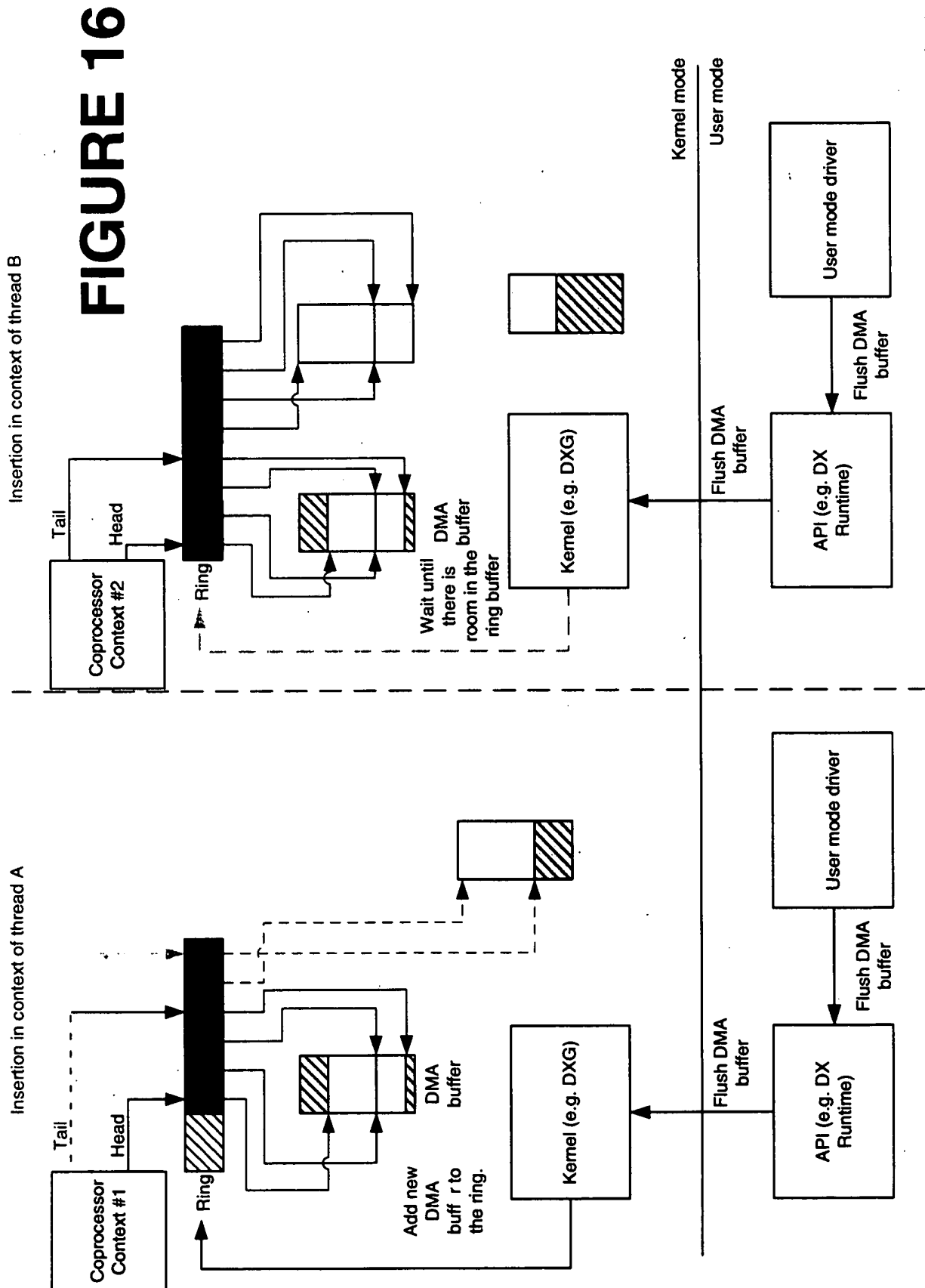


FIGURE 15



PROCESS A: Submit (IRQL passive, rendering thread context, coprocessor Context mutex held)

Acquire the VIDMM_lock.
Process the list of resources given, and update the usage information about allocations in this process.
Release the VIDMM lock.
Take the scheduler lock.
Call the driver to insert the current DMA buffer into the ring.
If the driver succeeded.
 If the context was idle.
 Insert the context back in the ready list at the tail of the queue for its current priority.
 If there is no context transfer pending and the current context is lower priority than the current context.
 Call the driver to context switch to this context.
 Signal that a context switch is pending.
Release the scheduler lock.
If the driver failed, the ring was full.
 Wait on an event that will be signaled when room becomes available.
 After the wait, go back to acquiring the scheduler lock.
If there is enough room left in the DMA buffer for another submission.
 Return to user mode with the current DMA buffer.
Acquire a new DMA buffer from the context's pool.
If DMA pool couldn't give another buffer at this time
 Wait on an event that will be signaled when a DMA buffer is inserted back into the pool.
When the wait is over, go back to trying to get a new DMA buffer.
Return the new DMA buffer to user mode.

PROCESS B: Context switch done (IRQL device, any thread context)

Take the scheduler lock.
 If a higher priority context is now ready for execution.
 Call the driver to context switch to the highest priority context.
 Else
 Signal that no context switch is currently pending.
Release the scheduler lock

PROCESS C: Quantum expires (IRQL device, any thread context)

Take the scheduler lock.
Reset the current priority of the context to its base priority.
Insert the context back at the end of the queue for its current priority.
If no context switches are currently pending.
 Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

FIGURE 17(A)

PROCESS D: Task finishes (IRQ device, any thread context)

Take the scheduler lock.
Ask the driver whether the context is really empty.
If the context is really empty.
 Reset the current priority of the context to its base priority.
 Insert the context in the idle list.
If the context wasn't really empty.
 If no context switches are currently pending.
 Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

PROCESS E: Page Fault (IRQ device, any thread context)

Take the scheduler lock.
Remove the context from the ready list.
Insert the context in the in page list as an atomic operation.
If in page thread currently sleeping.
 Queue a DPC to signal to wakeup the worker thread.
If no context switch are currently pending.
 Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

PROCESS F: Fault resolved (IRQ device, any thread context)

Take the scheduler lock.
Remove the context from the in page list.
Insert the context back in the ready list for its current priority.
If co context switches are currently pending, and the current context is higher priority than the currently running context.
 Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

FIGURE 17(B)

PROCESS G: In page worker thread

Go through the list of contexts in the inpage queue. Pick up the highest priority one.
Ask the driver for the list of resources required to make forward progress on the context.
Take the VIDMM lock.
Find a location for each of the allocations required for forward progress.
Invalidate the virtual address or handle for the allocation getting evicted.
Ask the driver to fill a DMA buffer with the memory transfer commands necessary to
bring the required allocations to their selected spots.
Release the VIDMM lock.
Submit the VidMm context as a regular coprocessor context.
If the list of contexts is empty, sleep until an item gets added.
Go back to the beginning of the loop.

PROCESS H: Periodic timer (passive level, system thread context)

Take the scheduler lock.
Increase the current priority of each context.
Release the scheduler lock.

FIGURE 17(C)

FIGURE 18

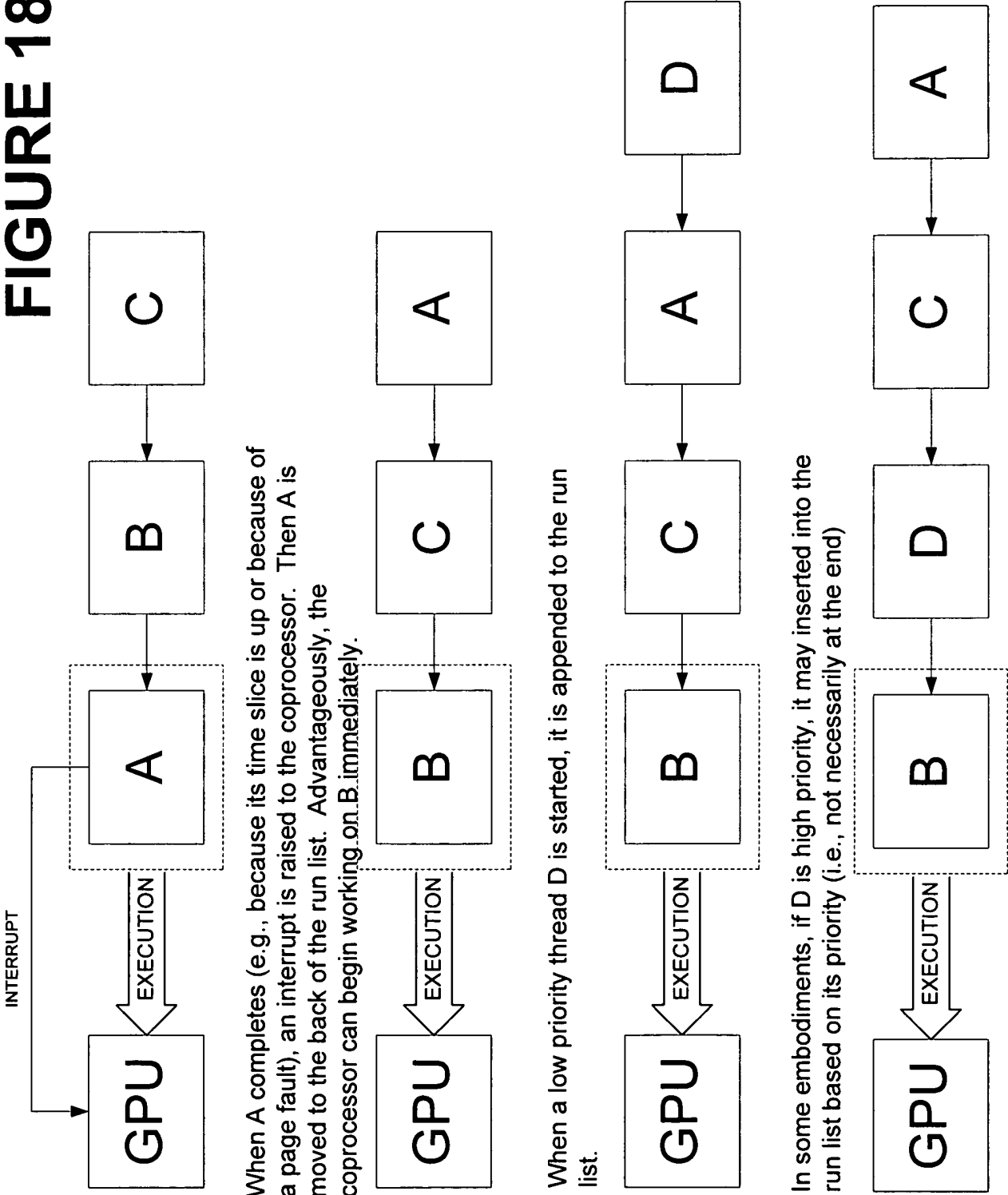
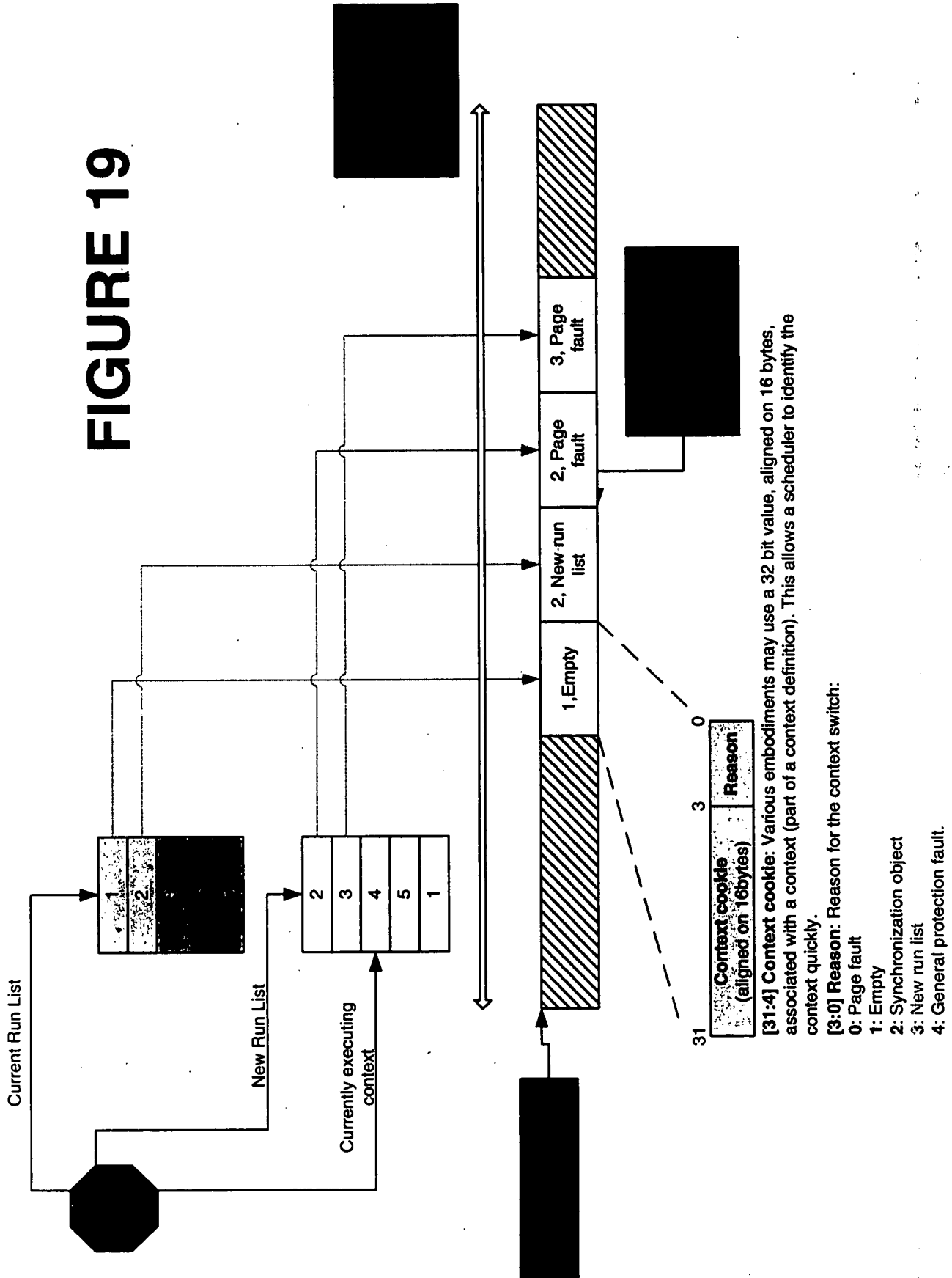


FIGURE 19



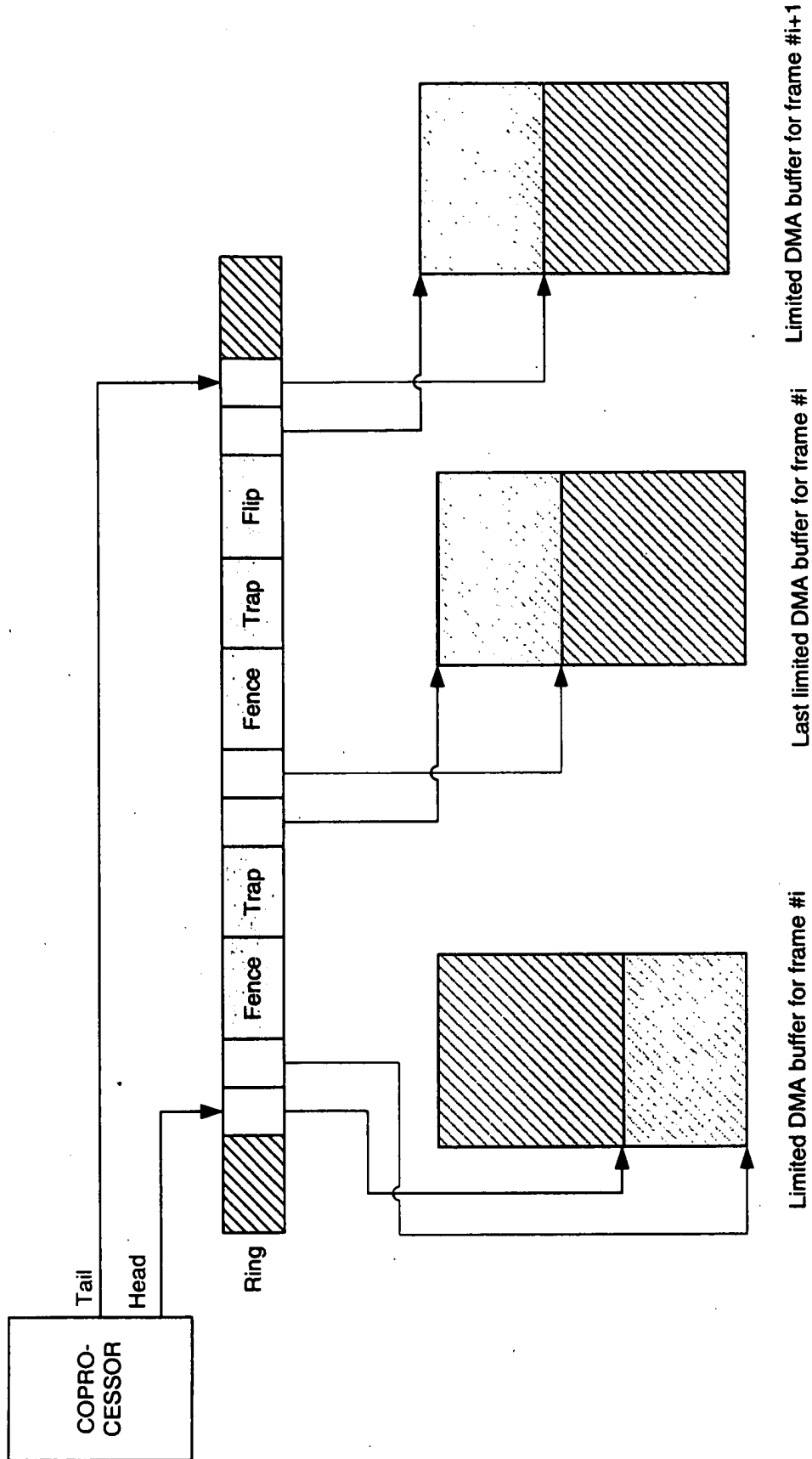


FIGURE 20

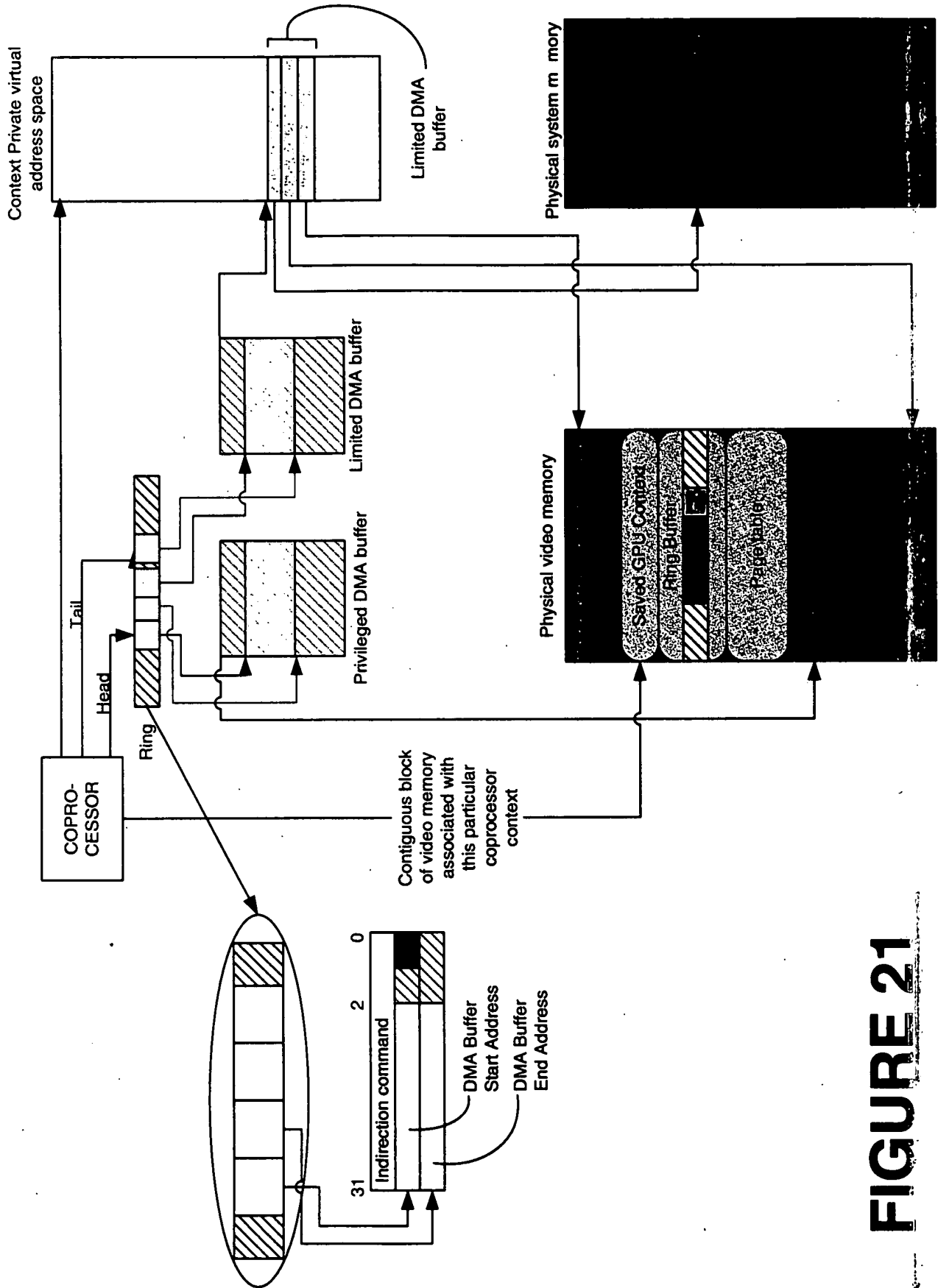


FIGURE 21

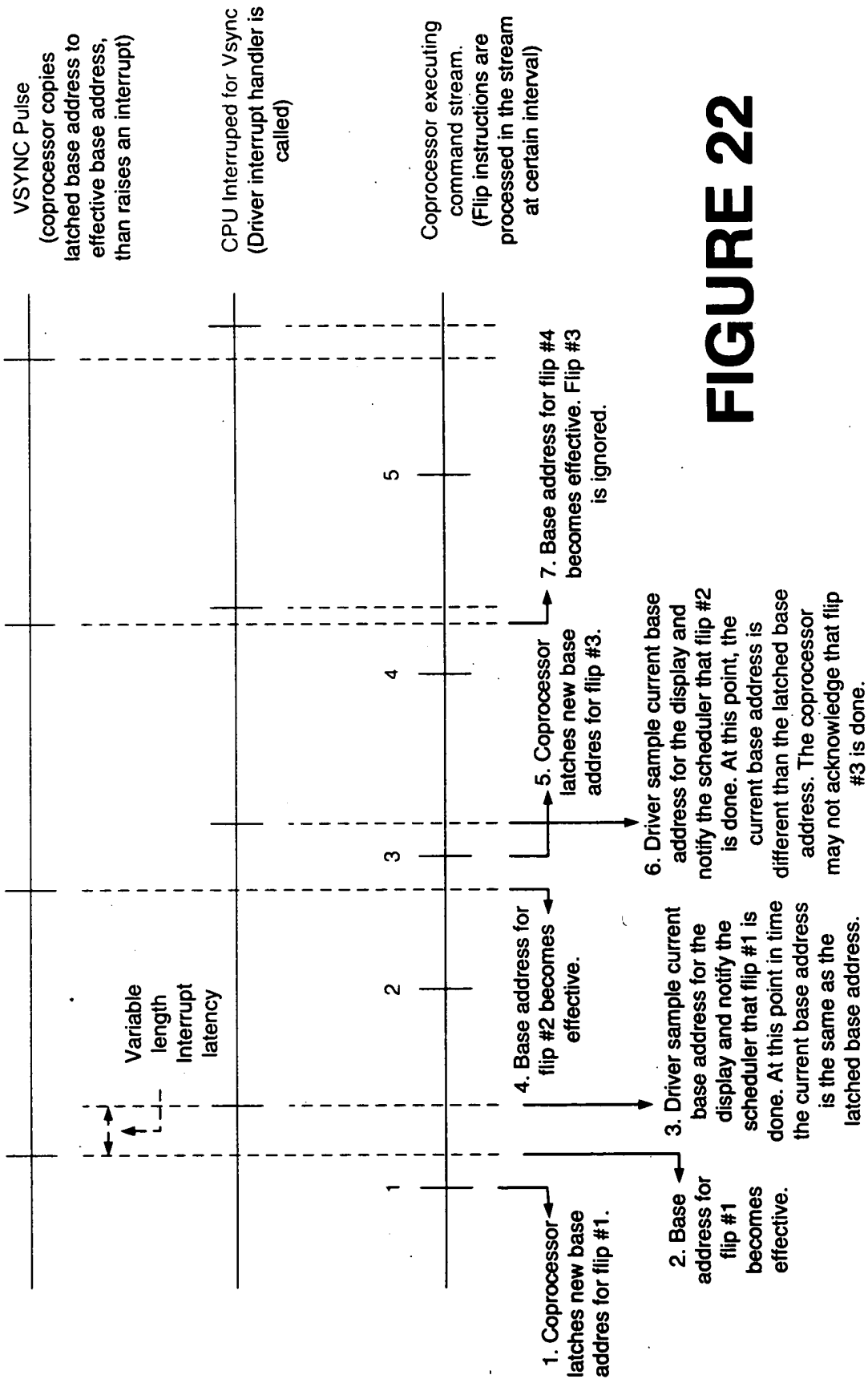


FIGURE 22

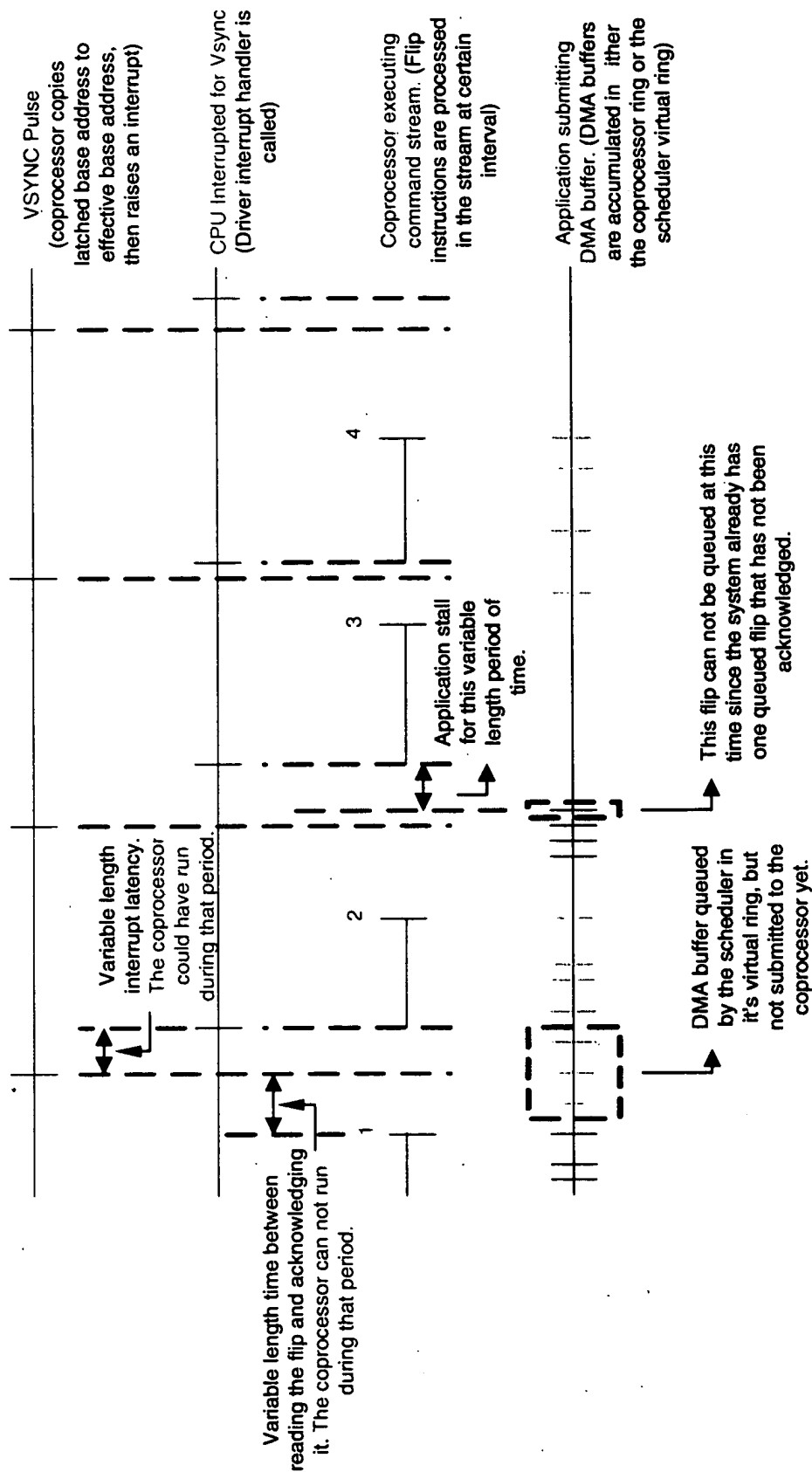
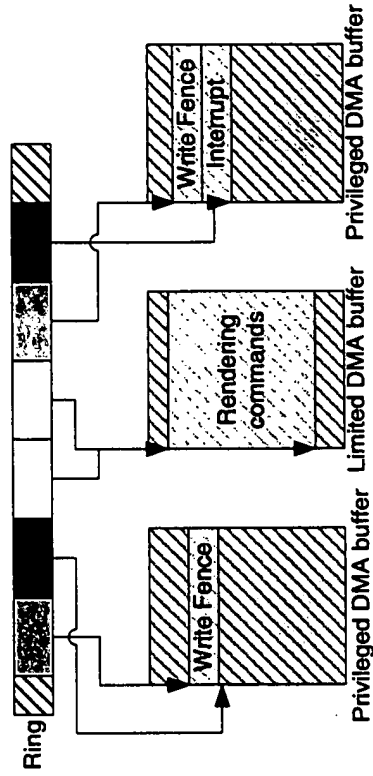


FIGURE 23

Coprocessor Thread B

Pseudo code:
 // Wait until we have exclusive access to the shared surface.
 //
 DxAcquireMutex(gSharedMutex);
 // Set the shared surface as a texture.
 //
 DxSetTexture(gSharedSurface);
 // Render what we need with the shared surface.
 //
 DxDrawSomething();
 // We're done with rendering, release the mutex.
 //
 DxReleaseMutex(gSharedMutex)

Coprocessor stream:



Coprocessor Thread A

Pseudo code:
 // Wait until we have exclusive access to the shared surface.
 //
 DxAcquireMutex(gSharedMutex);
 // Set the shared surface as the render target.
 //
 DxSetRenderTarget(gSharedSurface);
 // Render what we need in the shared surface.
 //
 DxDrawSomething();
 // We're done with rendering, release the mutex.
 //
 DxReleaseMutex(gSharedMutex)

Coprocessor stream:

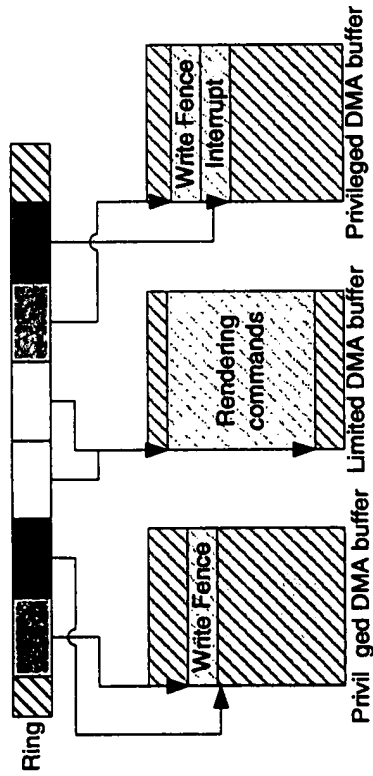


FIGURE 24

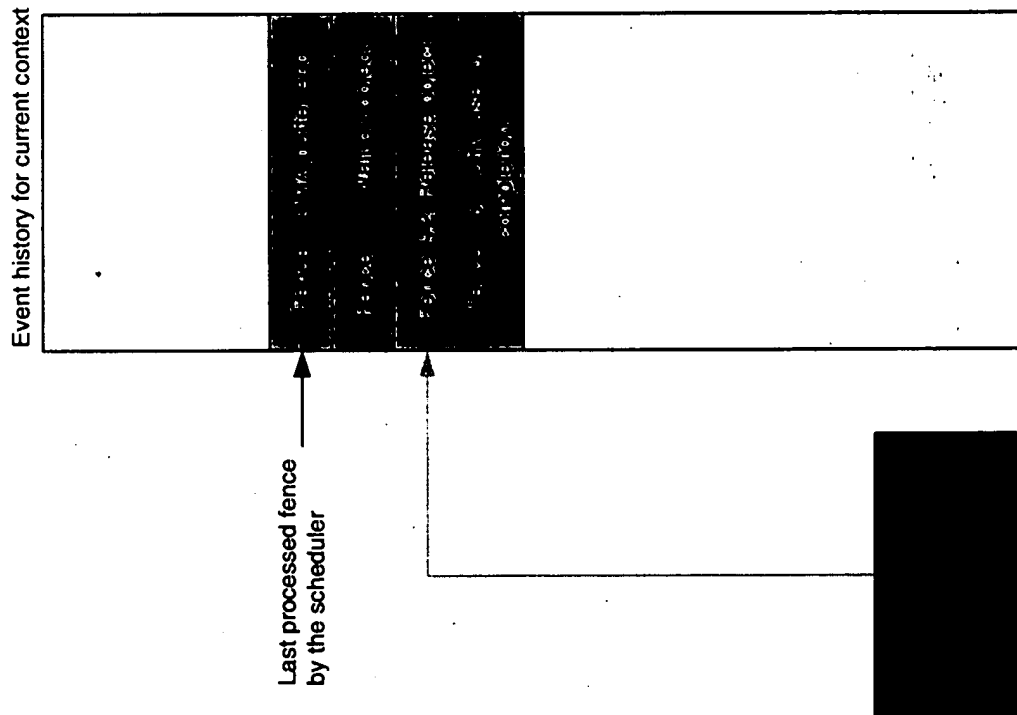
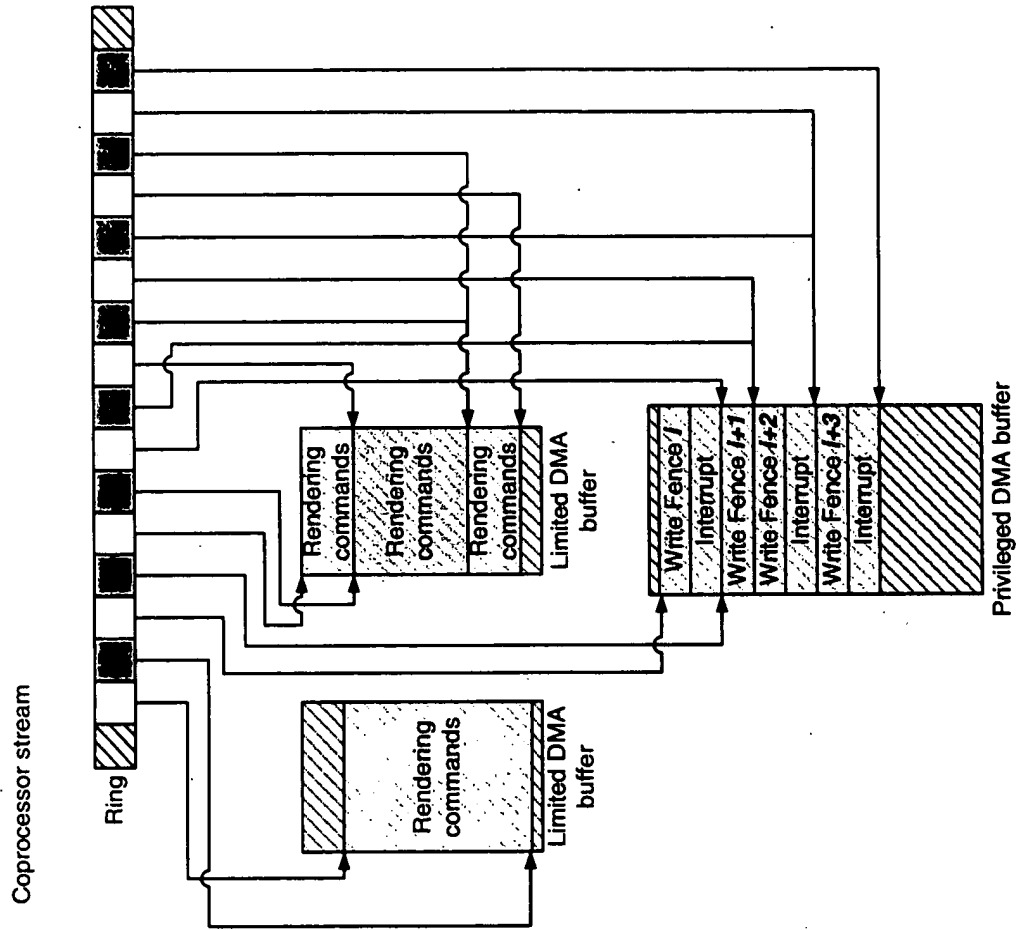


FIGURE 25

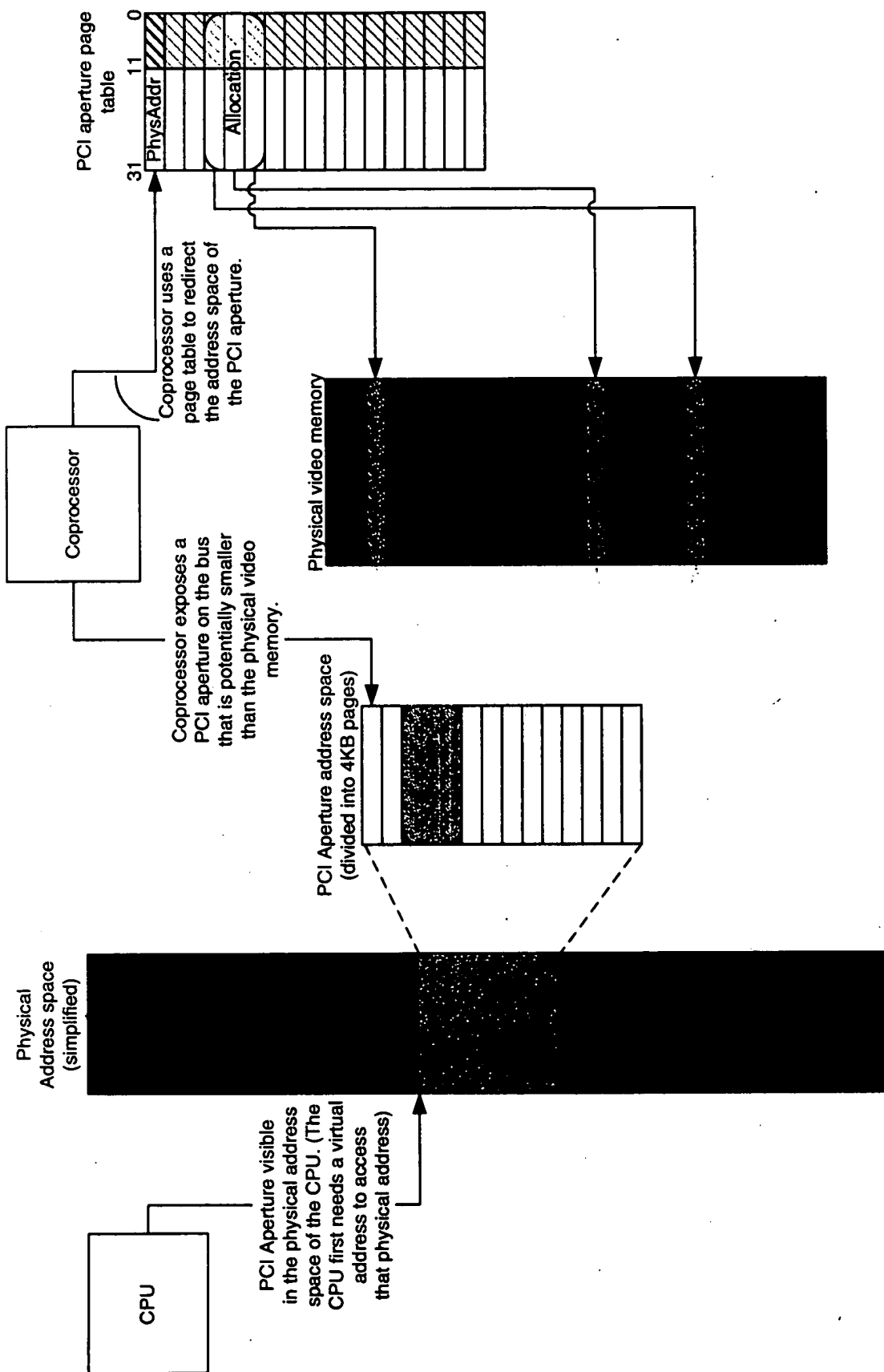


FIGURE 26